Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT

Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos*

Department of Computer Science
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
mail@fhutter.de,{davet,hoos}@cs.ubc.ca
WWW home page: http://www.cs.ubc.ca/labs/beta

Abstract. In this paper, we study the approach of dynamic local search for the SAT problem. We focus on the recent and promising Exponentiated Sub-Gradient (ESG) algorithm, and examine the factors determining the time complexity of its search steps. Based on the insights gained from our analysis, we developed Scaling and Probabilistic Smoothing (SAPS), an efficient SAT algorithm that is conceptually closely related to ESG. We also introduce a reactive version of SAPS (RSAPS) that adaptively tunes one of the algorithm's important parameters. We show that for a broad range of standard benchmark problems for SAT, SAPS and RSAPS achieve significantly better performance than both ESG and the state-of-the-art WalkSAT variant, Novelty$^+$.

1 Introduction and Background

The Satisfiability problem (SAT) is an important subject of study in many areas of computer science. Since SAT is $\mathcal{NP}$-complete, there is little hope to develop a complete algorithm that scales well on all types of problem instances; however, fast algorithms are needed to solve big problems from various domains, including prominent AI problems such as planning [7] and constraint satisfaction [2]. Throughout this paper, we focus on the model finding variant of SAT: Given a propositional formula F, find a model of F, *i.e.*, an assignment of truth values to the propositional variables in F under which F becomes true. As with most other work on SAT algorithms, we consider only propositional formulae in conjunctive normal form (CNF), *i.e.*, formulae of the form $F = \bigwedge_i \bigvee_j l_{ij}$, where each l_{ij} is a propositional variable or its negation. The l_{ij} are called *literals*, while the disjunctions $\bigvee_j l_{ij}$ are called *clauses* of F.

Some of the best known methods for solving SAT are Stochastic Local Search (SLS) algorithms; these are typically incomplete, *i.e.*, they cannot determine with certainty that a given formula is unsatisfiable, but they often find models of satisfiable formulae surprisingly effectively [5]. Although SLS algorithms for SAT differ in their implementation details, the general search strategy is mostly

* Corresponding author

the same [2]. Starting from an initial, complete assignment of truth values to all variables in the given formula F, in each search step, the truth assignment of one variable is changed from true to false or vice versa; this type of search step is also called a *variable flip*. Since the models of F are characterised by the fact that they leave none of F's clauses unsatisfied, variable flips are typically performed with the purpose of minimising an objective function that maps any variable assignment x to the number of clauses unsatisfied under x.

Since the introduction of GSAT [14], a simple best-improvement search algorithm for SAT, much research has been conducted in this area. Major performance improvements were achieved by the usage of noise strategies [12] and the development of the WalkSAT architecture [13]. In each search step, WalkSAT algorithms first choose a currently unsatisfied clause and then flip a variable occurring in this clause. Extensive experiments resulted in the introduction of sophisticated schemes for selecting the variable to be flipped, including the well-known Novelty and R-Novelty algorithms [8]. Further insight into the nature and theoretical properties of these algorithms motivated the more recent Novelty$^+$ variant [3], which is amongst the state-of-the-art algorithms for SAT.

In parallel to the development of more refined versions of randomised iterative improvement strategies like WalkSAT, another SLS method has become increasingly popular in SAT solving. This method is based on the idea of modifying the evaluation function in order to prevent the search from getting stuck in local minima or other attractive non-solution areas of the underlying search space. We call this approach Dynamic Local Search (DLS). DLS strategies for SAT typically associate weights with the clauses of the given formula, which are modified during the search process. These algorithms then try to minimise the total weight rather than the number of the unsatisfied clauses. GSAT with clause weighting [12] was one of the first algorithms based on this idea, although it changes weights only in connection with restarting the search process. Many variants of this scheme have been proposed: Frank [1] uses a DLS weighting scheme that is updated every time a variable is flipped. Morris' Breakout Method [9] simply adds one to the weight of every unsatisfied clause whenever a local minimum is encountered. The Discrete Lagrangian Method (DLM) [15] is based on a tabu search procedure and uses a similar, but slightly more complicated weight update scheme. Additionally, DLM periodically and deterministically envokes a smoothing mechanism that decreases all clause weights by a constant amount. The Smoothed Descent and Flood (SDF) approach [10] introduced a more complex smoothing method, and the concept of multiplicative weight updates. The most recent and best-performing DLS algorithm for SAT is the Exponentiated Sub-Gradient (ESG) method [11]. ESG, described in more detail in the next section, reaches or exceeds the performance of the best known WalkSAT algorithms in many cases.

In this paper we introduce "Scaling and Probabilistic Smoothing" (SAPS), a new algorithm that is conceptually closely related to ESG, but differs in the way it implements weight updates: SAPS performs computationally expensive weight smoothing probabilistically and less frequently than ESG. This leads

to a substantial reduction in the time complexity of the weight update procedure without increasing the number of variable flips required for solving a given SAT instance. Furthermore, different from ESG, SAPS can be implemented efficiently in a rather straight-forward way. We also introduce RSAPS, a partially self-tuning variant of SAPS that robustly reaches and in some cases exceeds the performance of SAPS with manually tuned parameters. As our empirical evaluation shows, SAPS and RSAPS outperform both ESG and Novelty$^+$, two of the best performing SLS algorithms for SAT, on a wide range of random and structured SAT instances, which suggests that these new algorithms might be the best SLS algorithms for SAT currently known.

The remainder of this paper is structured as follows. In Section 2 we review the ESG algorithm and discuss some of its important characteristics. Based on these insights, we present our new SAPS algorithm, a variant of the ESG approach, in Section 3. A self-tuning variant of this algorithm, RSAPS, is introduced in Section 4. In Section 5, we report results from our empirical study of SAPS and RSAPS which illustrate the performance improvements these algorithms achieve as compared to ESG and Novelty$^+$. Finally, Section 6 contains conclusions and points out directions for future work.

2 The ESG algorithm

The Exponentiated Subgradient (ESG) algorithm by Schuurmans, Southey, and Holte [11] is motivated by established methods in the operations research literature. Subgradient optimisation is a method for minimising Lagrangian functions that is often used for generating good lower bounds for branch and bound techniques or as a heuristic in incomplete local search algorithms.

ESG for SAT works as follows: The search is initialised by randomly chosen truth values for all propositional variables in the input formula, F, and by setting the weight associated with each clause in F to one. Then, in each iteration, a weighted search phase followed by a weight update is performed.

The weighted search phase consists of a series of greedy variable flips ("primal search steps"); in each of these, a variable is selected at random from the set of all variables that appear in currently unsatisfied clauses and when flipped, lead to a maximal reduction in the total weight of unsatisfied clauses. When reaching a local minimum state, *i.e.*, an assignment in which flipping any variable that appears in an unsatisfied clause would not lead to a decrease in the total weight of unsatisfied clauses, with probability η, the search is continued by flipping a variable that is uniformly chosen at random from the set of all variables appearing in unsatisfied clauses. Otherwise, the weighted search phase is terminated.

After each weighted search phase, the clause weights are updated ("dual search step"). This involves two stages: First, the weights of all clauses are multiplied by a factor depending on their satisfaction status; weights of satisfied clauses are multiplied by α_{sat}, weights of unsatisfied clauses by α_{unsat} (scaling stage). Then, all clause weights are pulled towards their mean value using the

formula $w \leftarrow w \cdot \rho + (1 - \rho) \cdot \bar{w}$ (smoothing stage), where $\bar{w}$ is the average of all clause weights after scaling, and the parameter ρ has a fixed value between zero and one. The algorithm terminates when a satisfying assignment for F has been found or when a maximal number of iterations has been performed. (For details, see [11].)

In a straight-forward implementation of ESG, the weight update steps ("dual search steps") are computationally much more expensive than the weighted search steps ("primal search steps"), whose cost is determined by the underlying basic local search procedure. Each weight update step requires accessing all clause weights, while a weighted search step only needs to access the weights of the critical clauses, *i.e.*, clauses that can change their satisfaction status when a variable appearing in a currently unsatisfied clause is flipped.[1] Typically, for the major part of the search only few clauses are unsatisfied; hence, only a small subset of the clauses is critical, rendering the weighted search steps computationally cheaper than weight updates.

If weight updates would typically occur very infrequently as compared to weighted search steps, the relatively high complexity of the weight update steps might not have a significant effect on the performance of the algorithm. However, experiments (not reported here) indicate that the fraction of weighting steps performed by ESG is quite high; it ranges from around 7% (for SAT encodings of large flat graph colouring problems) to more than 40% percent (for SAT-encoded all-interval-series problems).

Efficient implementations of ESG therefore critically depend on additional techniques in order to reach the competitive performance results reported in [11]. The most recent publically available ESG-SAT software by Southey and Schuurmans (Version 1.4), for instance, uses $\alpha_{sat} = 1$ (which avoids the effort of scaling satisfied clauses), replaces $\bar{w}$ by 1 in the smoothing step, and utilises a lazy weight update technique which updates clause weights only when they are needed. In Table 1, we compare this algorithm with the WalkSAT variant Novelty$^+$. Especially the step performance of ESG is quite impressive for a variety of problem instances; while it never needs more variable flips, sometimes it outperforms Novelty$^+$ by more than an order of magnitude. In most cases, ESG's time performance is still somewhat better than that of Novelty$^+$, but even with the optimisations in Version 1.4, ESG-SAT does not always reach the performance of Novelty$^+$ in terms of CPU time. Hence, it seems that the complexity of the weight update steps severely limits the performance of ESG in particular and dynamic local search algorithms for SAT in general.

3 Scaling and Probabilistic Smoothing (SAPS)

Based on the observations from the previous section, the most obvious way to improve the performance of ESG would be to reduce the complexity of the weight update procedure while retaining the relatively low number of weighted search

[1] The complexity of all other operations is dominated by these operations.

Problem	Novelty$^+$			ESG					
Instance	noise	steps	time	α	ρ	noise	pr. steps	d. steps	time
bw_large.a	0.40	7,007	**0.014**	3.0	0.995	0.0015	2,445	282	0.016
bw_large.b	0.35	125,341	0.339	1.4	0.99	0.0005	26,978	4,612	**0.280**
bw_large.c	0.20	3,997,095	**16.0**	1.4	0.99	0.0005	1,432,003	193,700	38.10
logistics.c	0.40	101,670	**0.226**	2.2	0.99	0.0025	9,714	4,664	0.229
flat100-med	0.55	7,632	**0.008**	1.1	0.99	0.0015	6,313	1,154	0.013
flat100-hard	0.60	84,019	0.089	1.1	0.99	0.0015	20,059	2,794	**0.037**
flat200-med	0.60	198,394	**0.208**	1.01	0.99	0.0025	96,585	7,587	0.237
flat200-hard	0.60	18147719	18.862	1.01	0.99	0.0025	2,511,228	213,995	**5.887**
uf100-hard	0.55	29,952	0.046	1.15	0.99	0.001	2,223	638	**0.006**
uf250-med	0.55	9,906	**0.015**	1.15	0.99	0.003	7,006	1,379	0.0195
uf250-hard	0.55	1,817,662	2.745	1.15	0.99	0.003	165,212	26,772	**0.461**
uf400-med	0.55	100,412	**0.160**	1.15	0.99	0.003	100,253	10,016	0.324
uf400-hard	0.55	14,419,948	22.3	1.15	0.99	0.003	3,015,013	282,760	**9.763**
ais10	0.40	1,332,225	4.22	1.9	0.999	0.0004	13,037	9,761	**0.139**

Table 1. Median number of steps and run-time on individual benchmark instances for ESG (Version 1.4) and Novelty$^+$; boldface indicates the CPU time of the faster algorithm. For all runs of Novelty$^+$, $wp = 0.01$, steps for ESG are split into primal and dual steps. Estimates for all instances are based on 100 runs. For details on the experimental methodology and the problem instances, see Section 5.

steps required to solve a given problem instance. As we will see in this section, this can be achieved in a rather simple and straight-forward way, leading to our new SAPS algorithm, a simple, yet efficient variant of ESG.

Two key observations provide the basis for the modified weight update scheme underlying SAPS. In the following we let C denote the set of all clauses of a given formula F and U_c the set of all clauses unsatisfied under the current variable assignment. We first note that the scaling operation can be restricted to the unsatisfied clause weights ($\alpha_{sat} = 1$) without affecting the variable selection in the weighted search phase, since rescaling all clause weights by a constant factor does not affect the variable selection mechanism. (As mentioned before, Southey's and Schuurmans' ESG implementation also makes use of this fact.) Based on our previous argument, this reduces the complexity of the scaling stage from $\theta(|C|)$ to $\theta(|U_c|)$. After a short initial search phase, $|U_c|$ becomes rather small compared to $|C|$; this effect seems to be more pronounced for larger SAT instances with many clauses. The smoothing stage, however, has complexity $\theta(|C|)$, and now dominates the complexity of the weight update.

Given this situation, the second key idea is to reduce the time complexity of the weight update procedure by performing the expensive smoothing operation only occasionally. Our experimental results show that this does not have a detrimental effect on the performance of the algorithm in terms of the number of weighted search steps required for solving a given instance. Towards the end of this section we will provide some intuition into this phenomenon.

```
procedure UpdateWeights(F, x, W, α, ρ,P_smooth)
  input:
    propositional formula F, variable assignment x, clause weights W = (w_i),
    scaling factor α, smoothing factor ρ, smoothing probability P_smooth
  output:
    clause weights W
  C = {clauses of F}
  U_c = {c ∈ C | c is unsatisfied under x}
  for each i s.t. c_i ∈ U_c do
    w_i := w_i × α
  end
  with probability P_smooth do
    for each i s.t. c_i ∈ C do
      w_i := w_i × ρ + (1 − ρ) × w̄
    end
  end
  return (W)
end
```

Fig. 1. The SAPS weight update procedure; $\bar{w}$ is the average over all clause weights.

Figure 1 shows our novel weight update procedure which is based on these insights. Different from the standard ESG weight update, this procedure scales the weights of unsatisfied clauses, but only smoothes all clause weights with a certain probability P_{smooth}. Thus, we call the corresponding algorithm *Scaling and Probabilistic Smoothing* (SAPS). Compared to ESG, in SAPS the complexity of *UpdateWeights* is reduced from $\Theta(|C|+|U_c|)$ to $\Theta(P_{smooth}\cdot|C|+|U_c|)$. As a result, the amortised cost of smoothing no longer dominates the algorithm's runtime. Obviously, there are other ways of achieving the same effect. For instance, similar to the mechanism found in DLM, smoothing could be performed deterministically after a fixed number of scaling stages. However, the probabilistic smoothing mechanism has the theoretical advantage of preventing the algorithm from getting trapped in cyclic behaviour (see also [3]). Furthermore, it is not clear that the possibility of performing smoothing should be restricted to situations where a local minimum of the evaluation function has been encountered. In fact, preliminary experimental results (not reported here) suggest that decoupling the smoothing operation from local minima results in an approximately optimal setting of P_{smooth} that is more stable over different domains.

Figure 2 shows the main SAPS algorithm and its underlying weighted search procedure; overall, this algorithm is conceptually very similar to ESG, except for the weight update procedure which has substantially smaller time complexity and provides the key for its excellent performance (see Section 5). The SAPS algorithm as described here does not require additional implementation tricks other than the standard mechanism for efficiently accessing critical clauses that is used in all efficient implementations of SLS algorithms for SAT. In particular, different from Southey's and Schuurmans' ESG implementation, SAPS does not replace $\bar{w}$ by one in the smoothing stage, or perform lazy weight updates.

```
procedure SAPS(F, α, ρ, wp, P_smooth)
  input:
    propositional formula F, scaling factor α,
    smoothing factor ρ, random walk probability wp,
    smoothing probability P_smooth
  output:
    variable assignment x or ∅
  x := Init(F)
  W := InitWeights(F)
  while not terminate(F, x) do
    x' := WeightedSearchStep(F, x, W)
    if x' = ∅ then
      with probability wp do
        x := RandomStep(F, x)
      otherwise
        W := UpdateWeights(F, x, W, α, ρ, P_smooth)
      end
    else
      x := x'
    end
  end
  if (F is not satisfied under x) then
    x = ∅
  end
  return (x)
end

procedure WeightedSearchStep(F, x, W)
  input:
    propositional formula F, variable assignment x, clause weights W
  output:
    variable assignment x̂ or ∅
  U_v = {variables of F that appear in clauses unsatisfied under x}
  X' := {x̂ | x̂ is x with variable v ∈ U_v flipped}
  best :=min{g(F, x̂, W) | x̂ ∈ X'}
  X := {x̂ ∈ X' | g(F, x̂, W) = best}
  if best ≥ 0 then
    x̂ := ∅
  else
    x̂ := draw(X)
  end
  return (x̂)
end
```

Fig. 2. The SAPS Algorithm. 'Init' randomly initialises x, 'InitWeights' initialises all clause weights to 1. 'RandomStep(F,x)' returns an assignment obtained from x by flipping a variable that has been selected uniformly at random from the set of all variables of F; and $g(F, \hat{x}, W)$ denotes the total weight of the clauses in F that are unsatisfied under assignment $\hat{x}$. The function 'draw(X)' returns an element that is uniformly drawn at random from set X.

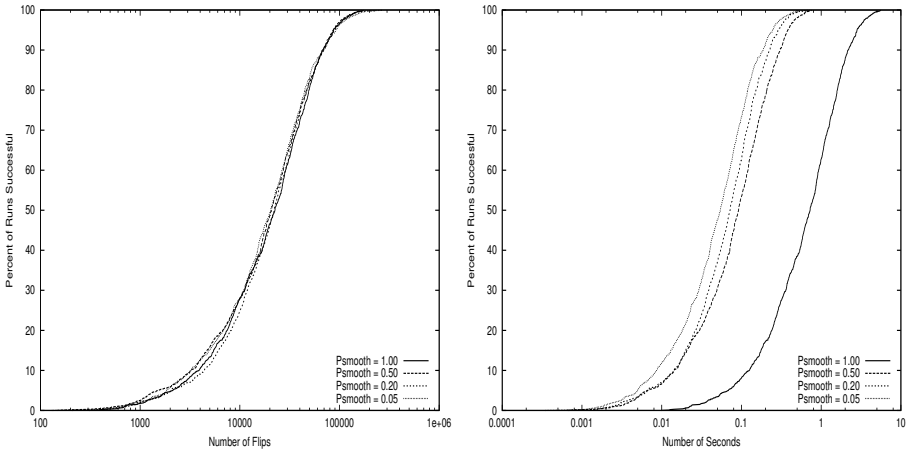

Fig. 3. Flip performance (left) and time performance (right) for SAPS with different values of P_{smooth} for problem instance `ais10`.

Figure 3 illustrates the effect of varying the smoothing probability, P_{smooth}, on the performance of SAPS, while simultaneously decreasing ρ to compensate for "missed" smoothing stages. Setting P_{smooth} to one results in an algorithm that is very closely related (but still not identical) to ESG. When decreasing P_{smooth} below one, we observe unchanged step performance while the time performance is improving. For some SAT instances, especially from the logistics and blocksworld planning domains, we achieve best time performance for $P_{smooth} = 0$, *i.e.*, when no smoothing is used at all; however, most instances require at least some smoothing. For our computational experiments, unless explicitly noted otherwise, we generally used $P_{smooth} = 0.05$, a setting which resulted in reasonable performance over a broad range of SAT instances. However, in many cases, $P_{smooth} = 0.05$ is clearly not the optimal setting; therefore, in the next section we introduce a scheme for automatically adapting the smoothing probability over the course of the search process.

To gain a deeper understanding of the performance of the ESG and SAPS algorithms and specifically the role of the parameters α, ρ and P_{smooth}, it is useful to study the evolution of clause weights over time. If two clauses were unsatisfied at only one local minimum each, then the relative weights of these clauses depend on the order in which they were unsatisfied. Since the weights are scaled back towards the clause weight average at each smoothing stage, the clause that has been unsatisfied more recently has a larger weight. So scaling and smoothing can be seen as a mechanism for ranking the clause weights based on search history. Clearly, the distribution of clause weights, which is controlled by the settings of α, ρ, and P_{smooth}, has a major impact on the variable selection underlying the primal search steps. Since uniform scaling of weights has no effect on variable selection and hence on the performance of the algorithm, we consider

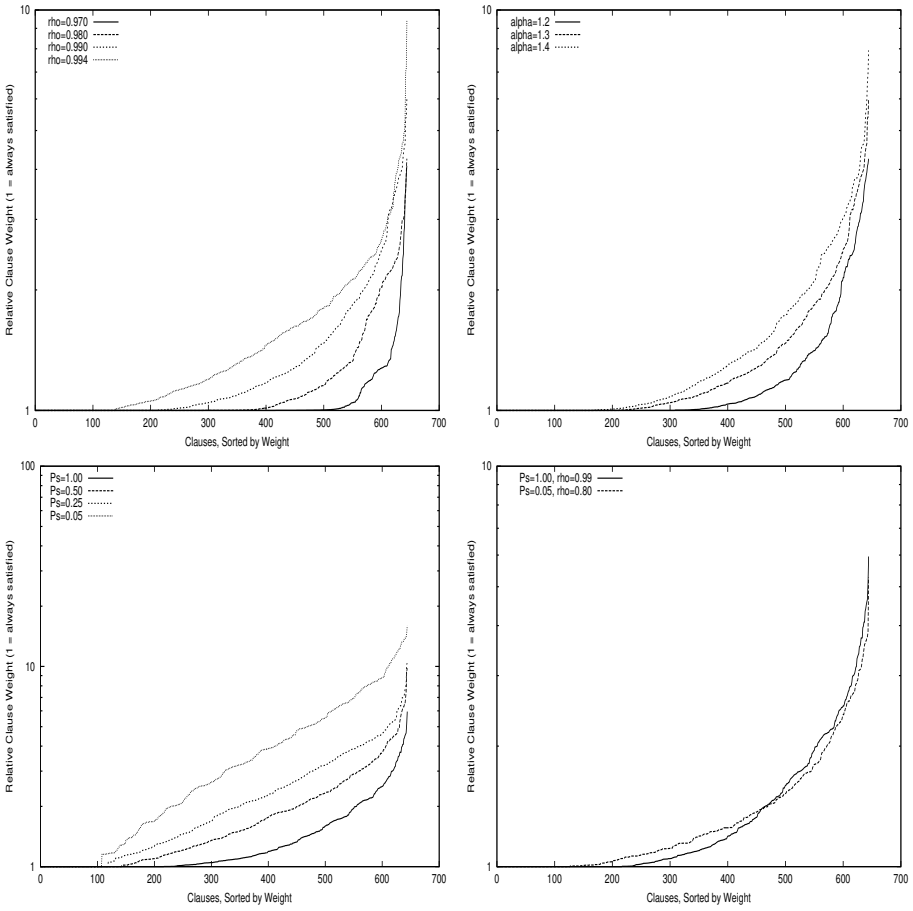

Fig. 4. Clause weight distribution (CWDs) for SAPS after 400 local minima for various values of ρ (top left), α (top right) and P_{smooth} (bottom left). Approximately identical CWDs are obtained for two different $(\alpha, \rho, P_{smooth})$ triplets for which SAPS shows similar step performance (bottom right). All CWDs are measured for Uniform Random 3-SAT instance `uf150-hard`. Unless otherwise noted, $\alpha = 1.3$, $\rho = 0.99$, $P_{smooth} = 1$, and $wp = 0$. All runs were initialised with the same random seed.

distributions over clause weights that are normalised by multiplying all weights by a constant factor such that clauses that have never been unsatisfied have an adjusted weight of one.

Figure 4 shows typical clause weight distributions (CWDs) for a given SAT instance, *i.e.* all clause weights sorted by weight, for different settings of α, ρ, and P_{smooth} after 400 local minima have been encountered. In our experience, after a certain number of local minima, the CWD converges to a specific distribution that is determined by the problem instance and the settings of α, ρ, P_{smooth}.

We hypothesise that the shape of the CWD for a given problem instance determines the performance of ESG and SAPS. In Figure 4, we can see directly the effect of changing the parameters. The smoothing parameter has a significant impact on the shape of the CWD. Intuitively, the basic weighted local search will place greater emphasis on satisfying and keeping satisfied the clauses with higher clause weights. For smaller values of ρ, *i.e.*, more smoothing, fewer clauses have high weights, leading to a greedier, more intensified search. Conversely, less smoothing leads to CWDs characteristic for a more diversified search. Interestingly, these effects of the CWDs on the underlying weighted search can be interpreted as that of a soft tabu mechanism on clauses, where clauses with higher weights are "more taboo", *i.e.*, likely to stay satisfied longer.

We also found that if two different $(\alpha, \rho, P_{smooth})$ triplets result in nearly identical CWDs, they will also yield nearly identical performance results. Two such triplets that achieve similar step performance are $(1.3, 0.99, 1.0)$ and $(1.3, 0.80, 0.05)$; as can be seen in Figure 4 (bottom right), the respective CWDs are very similar. In the context of our earlier soft tabu interpretation, this suggests that a clause-based soft tabu algorithm imposing a SAPS- or ESG-like CWD could match the step performance of SAPS and ESG.

4 Reactive SAPS (RSAPS)

As mentioned before, the performance of SAPS depends on the smoothing probability, P_{smooth}, in addition to the three other parameters common to ESG and SAPS. Although we found that, as a rule of thumb, the settings $\alpha = 1.3$, $\rho = 0.8$, $wp = 0.01$, and $P_{smooth} = 0.05$ work reasonably robustly in many cases, there are better parameter settings for almost all problem instances tested here. Determining these settings manually can be difficult and time-consuming; therefore, it would be desirable to automatically find them during the search. In the following, we use a scheme analogous to the one recently applied by Hoos to automatically tuning the noise parameter of Novelty$^+$ [4]. The basic idea is to reactively use higher noise levels, leading to more search diversification, if and only if there is evidence for search stagnation, *e.g.* as a result of getting trapped in a local minimum region. Thus, if search stagnation is detected, more noise is introduced; otherwise, the noise value is gradually decreased.

The SAPS algorithm escapes from local minima by scaling the weights of unsatisfied clauses, whereas smoothing the weights back towards uniform values acts as an intensification of the search; complete smoothing ($\rho = 0$) results in basic GSAT behaviour without noise. This suggests that search intensification can be controlled reactively by either adapting ρ or P_{smooth}. At this stage, we neither considered adapting wp nor α, since using fixed values of 0.01 and 1.3, respectively, resulted uniformly and robustly in maximal performance of SAPS in most of our experiments.

Intuitively, it makes much sense to adapt the amount of smoothing since this directly determines the actual extent of search intensification. In order to let changes in ρ effectively control the search, the smoothing probability would

have to be rather high. However, in the previous section, we have seen that in order to achieve superior time performance, we need at least a bias towards low smoothing probabilities. Therefore, we decided to use a fixed value of ρ and to control the amount of smoothing by adapting P_{smooth}. By choosing a rather low value for ρ, large amounts of smoothing and high levels of search intensification can still be achieved, while keeping the average smoothing probability low.

It is important to realise the possible gain of adapting the smoothing probability during the search. Besides the obvious advantage of eliminating the need to manually tune P_{smooth}, because of the interdependence of P_{smooth} and ρ, an effective adaptive mechanism for the former parameter should be able to at least partly compensate for suboptimal settings of the latter. Furthermore, when smoothing is performed only in situations where it is actually required, in principle it is possible to obtain better performance than for *optimal* fixed settings of P_{smooth} and ρ. This is due to the fact that optimal settings may differ troughout the course of the search.

Our stagnation criterion is the same as used in Adaptive Novelty$^+$ [4]: if the search has not progressed in terms of a reduction in the number of unsatisfied clauses over the last $\textit{(number of clauses)} \cdot \theta$ variable flips, the smoothing probability is reduced, $\theta = 1/6$ seems to give uniformly good performance. Just like an increase of the noise value in Adaptive Novelty$^+$, this reduction of the smoothing probability leads to a diversification of the search. As soon as the number of unsatisfied clauses is reduced below its value at the last change of the smoothing probability, P_{smooth} is increased in order to intensify exploration of the current region of the search space. The exact mechanism for adapting P_{smooth} is shown in Figure 5. A bias towards low smoothing probabilities is achieved by decreasing P_{smooth} faster than increasing it. Moreover, after each smoothing operation, P_{smooth} is set to zero (this happens in procedure *UpdateWeights*). Together, these two mechanisms help to ensure low average values of P_{smooth} for problem instances that do not benefit from smoothing.

5 Experiments and Results

In order to evaluate the performance of SAPS and RSAPS against ESG as well as Novelty$^+$, we conducted extensive computational experiments on widely used benchmark instances for SAT obtained from SATLIB [6].[2] The benchmark set used for our evaluation comprises SAT-encoded blocksworld and logistics planning instances, SAT-encoded flat graph colouring problems, critically constrained Uniform Random-3-SAT instances, and SAT-encoded all-interval-series problems. To better assess scaling behaviour, we also used a recently generated test-set of 100 critically constrained, satisfiable Uniform Random-3-SAT instances with 400 variables and 1700 clauses each; this is the same set used in [4]. The instances labelled `*-hard` and `*-med` are those instances from the respective test-sets with maximal and median local search cost (*lsc*) for WalkSAT

[2] These instances can be found at http://www.satlib.org.

```
procedure AdaptSmoothingProbability(F, H, P_smooth)
    input:
        propositional formula F, partial search history H,
        smoothing probability P_smooth
    output:
        smoothing probability P_smooth
    C = {clauses of F}
    θ := 1/6; δ := 0.1
    if (no improvement has been made for |C| · θ steps) then
        P_smooth := δ × P_smooth
        mark the current step as the last improvement
    else if (an improvement has been made in this step) then
        P_smooth := P_smooth + 2δ(1 − P_smooth)
        mark the current step as the last improvement
    end
    return (P_smooth)
end
```

Fig. 5. Procedure for automatically adapting the smoothing probability P_{smooth}; RSAPS calls this procedure after every search step.

using manually tuned static noise and median *lsc*, respectively (again, these are the same instances as used in [4]).

All computational experiments reported here were executed on a dual 1GHz Pentium III PC with 256KB cache and 1GB RAM, running Red Hat Linux (Version 2.4.9-6smp). Computing times were measured and are reported in CPU seconds. For each problem instance and algorithm, we obtain empirical run-length and run-time distributions (RLDs and RTDs) [5] by solving the problem instance at least 100 times; the cutoff parameter was set to ∞.

In Table 2, for single problem instances from different domains, we present the medians of the RTDs obtained by SAPS and RSAPS. Generally, SAPS and RSAPS achieve superior performance over ESG and Novelty$^+$. In terms of number of search steps, SAPS is performing similar to ESG, with a slight advantage on larger problem instances. Due to the reduced complexity of smoothing, SAPS is outperforming ESG to a factor of up to six (`logistics.c`) in terms of CPU time. For smaller instances, such as `uf100-hard`, the time complexity is roughly the same; SAPS is never slower than ESG. Furthermore, for all problem instances cited in [11] where DLM outperformed ESG, SAPS and RSAPS outperform ESG by a greater margin.

When comparing SAPS to Novelty$^+$, the performance differences are more apparent and the time performance of SAPS is often more than an order of magnitude superior. The big blocksworld planning instance `bw_large.c` is the only case where Novelty$^+$ performs better than SAPS; RSAPS, however, achieves significantly better performance than Novelty$^+$ for this instance. We have evidence, however, that for the DIMACS graph colouring instances `g125.17` and `g125.18`, Novelty$^+$ performs substantially better than any of the dynamic local

Problem		SAPS, $P_{smooth} = 0.05$					RSAPS				
Instance	α	ρ	pr. steps	d. steps	time	sf	ρ	pr. steps	d. steps	time	
bw_large.a	1.3	0.8	2,233	331	**0.009**	1.56	0.8	2,413	306	**0.008**	
bw_large.b	1.3	0.8	29,452	3,205	**0.179**	1.56	0.8	25,392	2,404	**0.140**	
bw_large.c	1.1	0.6	1,866,748	264,211	37.88	0.42	0.9	1,472,480	138,235	**12.66**	
logistics.c	1.3	0.9	6,493	2,223	**0.037**	6.10	0.9	6,409	1,077	**0.030**	
flat100-med	1.3	0.4	5,437	1,118	0.008	1.00	0.4	6,367	1,292	0.010	
flat100-hard	1.3	0.8	22,147	3,501	**0.032**	1.16	0.8	19,627	2,837	**0.029**	
flat200-med	1.3	0.4	55,238	4,693	**0.087**	2.39	0.4	71,967	6,183	0.122	
flat200-hard	1.3	0.4	1,954,164	215,716	**3.052**	1.93	0.4	3,129,337	308,756	5.162	
uf100-hard	1.3	0.8	2,967	811	0.006	1.00	0.8	2,788	865	0.006	
uf250-med	1.3	0.4	5,445	1,159	**0.011**	1.36	0.4	5,302	1,253	0.012	
uf250-hard	1.3	0.7	144,021	26,348	**0.291**	1.58	0.7	118,960	21,346	**0.249**	
uf400-med	1.3	0.7	47,475	5,502	**0.103**	1.55	0.7	46,762	5,579	0.106	
uf400-hard	1.3	0.2	900,501	133,267	**1.973**	4.95	0.2	986,621	126,301	2.216	
ais10	1.3	0.9	13,482	6,449	**0.051**	2.73	0.9	12,491	6,916	**0.044**	

Table 2. Median run-time and number of steps on individual benchmark instances for our Dynamic Local Search approaches. For all runs, $wp = 0.01$. Estimates are based on 100 runs. Search steps are divided into primal search steps and dual search steps. For SAPS, bold face indicates superior time performance over both ESG and Novelty$^+$, for RSAPS, bold face indicates superior time performance over all the other algorithms. "sf" (speedup factor) denotes the time taken by the faster algorithm of ESG and Novelty$^+$ for the respective instance divided by the time taken by SAPS.

search algorithms. SAPS reduces the performance difference but seems unable to outperform Novelty$^+$.

The fact that in many cases, SAPS shows an improvement in step performance over ESG should be emphasised. We attribute this to problems arising from approximations used in efficient implementations of ESG. However, another possible explanation could lie in the additional randomness of SAPS. Since the clause weights in ESG are real numbers, this algorithm becomes almost deterministic after an initial search phase and can be trapped in a cycle, from which it can only escape by means of a random walk step performed in a local minimum.

In Section 3, we showed how the time complexity of smoothing increases linearly with problem size. This suggests that performance differences between ESG and SAPS should also increase with problem size. To avoid complications arising from different implementations, we use a variant of SAPS with $P_{smooth} = 1$ to illustrate these differences in scaling behaviour.[3] We refer to this variant as SAPS[1] and to regular SAPS as SAPS[0.05]. We demonstrate different scaling properties in time complexity *w.r.t.* problem size for the test sets `uf100` and `uf400`, both of which contain 100 Uniform Random 3-SAT formulae with 100 and 400 variables, respectively. In Figure 6 (top), we compare SAPS[1] with Novelty$^+$. We see impressive results for SAPS[1] on test-set `uf100`, whereas its

[3] The performance of this variant is very similar to ESG for small instances and seems marginally better for larger instances.

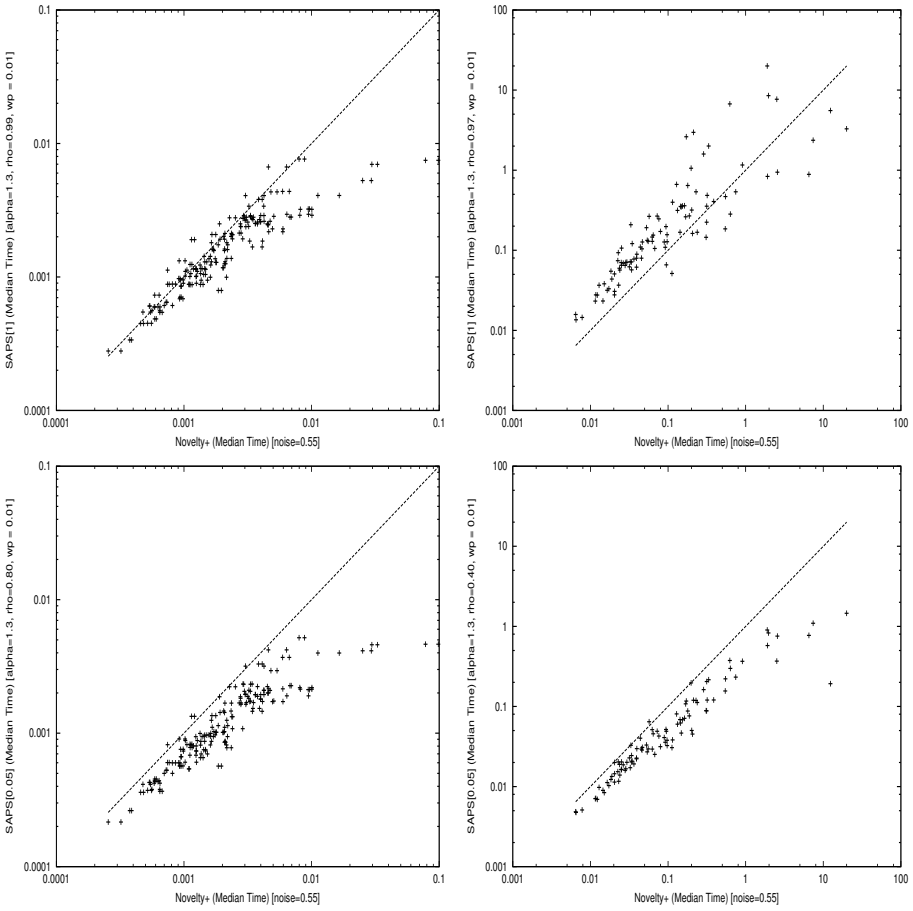

Fig. 6. Correlation between time performance of SAPS and Novelty$^+$ on Uniform Random 3-SAT test-sets uf100 (left), and uf400 (right); top: Novelty$^+$ *vs.* SAPS[1], bottom: Novelty$^+$ *vs.* SAPS[0.05].

performance degrades for the instances in test-set `uf400`. Next, we performed the same comparison for SAPS[0.05] and Novelty$^+$. As can be seen in Figure 6 (bottom), the difference to Novelty$^+$ for test-set `uf100` is about the same as for SAPS[1] and Novelty$^+$. However, for the larger instances in `uf400` it becomes obvious that the scaling behaviour of SAPS[0.05] is far superior to SAPS[1].

6 Conclusions & Future Work

As we have shown in this work, new insights into the factors underlying the runtime behaviour of the recent and promising Exponentiated Sub-Gradient (ESG) algorithm can lead to variants of this algorithm that show significantly improved

performance over both ESG as well as the best known WalkSAT algorithms and hence can be counted amongst the best performing SAT algorithms known to date. Furthermore, reactive search techniques can be used to reduce the need for manual parameter tuning of the resulting algorithms.

In future work, we plan to further investigate the role of the scaling and smoothing stages for ESG and SAPS. By studying clause weight distributions we hope to be able to better understand how these mechanisms interact with the basic weighted search algorithm underlying both algorithms. Ultimately, we are confident that this should lead to further performance improvements. It might even be possible to obtain such improvements with algorithms that are conceptually simpler than ESG or SAPS. Furthermore, it would be interesting to develop completely self-tuning variants of SAPS, which reactively adapt α and ρ as well as the smoothing probability P_{smooth}.

References

1. J. Frank. Learning Short-term Clause Weights for GSAT. In *Proc. IJCAI-97*, pp. 384–389, Morgan Kaufmann Publishers, 1997.
2. H.H. Hoos. *Stochastic Local Search — Methods, Models, Applications*, PhD thesis, Darmstadt University of Technology, 1998.
3. H.H. Hoos. On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. In *Proc. AAAI-99*, pp. 661–666. AAAI Press, 1999.
4. H.H. Hoos. An Adaptive Noise Mechanism for WalkSAT. To appear in *Proc.AAAI-02*, AAAI Press, 2002.
5. H.H. Hoos and T. Stützle. Local Search Algorithms for SAT: An Empirical Evaluation. In *J. of Automated Reasoning*, Vol. 24, No. 4, pp. 421–481, 2000.
6. H.H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In I.P. Gent, H. Maaren, T. Walsh (ed.), *SAT 2000*, pp. 283–292, IOS Press, 2000.
7. H. Kautz and B. Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proc. AAAI-96*, pp. 1194–1201. AAAI Press, 1996.
8. D.A. McAllester and B. Selman and H.A. Kautz. Evidence for Invariants in Local Search. In *Proc. AAAI-97*, pp. 321–326, AAAI Press, 1997.
9. P. Morris. The breakout method for escaping from local minima. In *Proc. AAAI-93*, pp. 40–45. AAAI Press, 1993.
10. D. Schuurmans, and F. Southey. Local search characteristics of incomplete SAT procedures. In *Proc. AAAI-2000*, pp. 297–302, AAAI Press, 2000.
11. D. Schuurmans, F. Southey, and R.C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proc. IJCAI-01*, pp. 334-341, Morgan Kaufmann Publishers, 2001.
12. B. Selman and H.A. Kautz. Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proc. IJCAI-93*, pp. 290–295, Morgan Kaufmann Publishers, 1993.
13. B. Selman and H.A. Kautz and B. Cohen. Noise Strategies for Improving Local Search. In *Proc. AAAI-94*, pp. 337–343, AAAI Press, 1994.
14. B. Selman, H. Levesque and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. AAAI-92*, pp. 440–446, AAAI Press, 1992.
15. Z. Wu and B.W. Wah. An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems. In *Proc. AAAI-00*, pp . 310–315, AAAI Press, 2000.